LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-213924

FY005 Accomplishments for Colony Project

Terry Jones, Laxmikant Kale, Jose Moreira

July 22, 2005

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Office of Science
U.S. DEPARTMENT OF ENERGY

Advanced Scientific Computing Research
Computer Science
FY 2005 Accomplishment

Colony – *"Services and Interfaces to Support Large Numbers of Processors"*

Terry Jones[1], Lawrence Livermore National Laboratory
Laxmikant Kale[2], University of Illinois at Urbana-Champaign
Jose Moreira[3], International Business Machines
Celso Mendes, University of Illinois at Urbana-Champaign
Sayantan Chakravorty, University of Illinois at Urbana-Champaign
Todd Inglett, International Business Machines
Andrew Tauferner, International Business Machines

Summary

The Colony Project is developing operating system and runtime system technology to enable efficient general purpose environments on tens of thousands of processors. To accomplish this, we are investigating memory management techniques, fault management strategies, and parallel resource management schemes. Recent results show promising findings for scalable strategies based on processor virtualization, in-memory checkpointing, and parallel aware modifications to full featured operating systems.

It is an ironic twist that one of the most beneficial trends in supercomputing is also one of its greatest challenges. While the trend towards larger processor counts benefits application developers through more processing power, it also challenges application developers to harness ever-increasing numbers of processors for productive work. Much of the burden falls to operating systems and runtime systems that were originally designed for much smaller processor counts. Under the Colony project, we are researching and developing system software to enable general purpose operating and runtime systems for tens of thousands of processors. These technologies will be demonstrated on multiple platforms including IBM's Blue Gene class machines.

As machines become larger, system-wide management of important resources across the entire becomes increasingly important for improving performance and scalability of applications. Important examples include CPU time and memory. Typically, application programmers have to explicitly write code to balance the use of these resources. However, with larger processor counts and more complex applications, programmers are faced with an increasingly difficult task in balancing resource usage. Next generation parallel operating and runtime systems must provide automatic resource balancing. We are developing, under Colony, an infrastructure and strategies for automated resource management by delegating low-level management tasks to operating system and runtime software. One impediment to automated resource management is the monolithic view of a process as an indivisible unit. As an alternative, we propose processor virtualization, in which a parallel application is divided into much

[1] Terry Jones, LLNL Principal Investigator, Telephone Number 925-423-9834, E-mail address trj@llnl.gov
[2] Laxmikant Kale, UIUC Principal Investigator, Telephone Number 217-244-0094, E-mail address kale@cs.uiuc.edu
[3] Jose Moreira, IBM Principal Investigator, Telephone Number 507-253-6383, E-mail address moreira@us.ibm.com

smaller migratable work units (objects) such as user-level threads and parallel objects. These relatively fine-grained units make it possible for the parallel operating system to manage resources automatically, and facilitate new fault-tolerance and job-management strategies. We have implemented this basic idea in Charm++ and in AMPI, an MPI implementation based on Charm++. This approach allows programs to be created more efficiently, using less programmer time and effort.

Without coordination in the memory management system across a parallel machine, concurrently allocated memory is represented by different virtual addresses on every task. By reserving and managing the range of virtual address space across a parallel machine at the operating system level, we expect to reduce the overhead and complexity of managing virtual memory in large parallel systems.

As the number of processors in a parallel system grows, the expected mean time between failure shrinks and fault tolerance becomes an important issue. We have now demonstrated the effectiveness of a sophisticated checkpoint/restart mechanism where the checkpoints were saved in memory, avoiding the wait for disk access. Further, we have developed techniques that enable the checkpoints to be created automatically, with no additional user code. Combined with automatic fault detection and recovery, this forms a complete fault recovery system. We are currently developing a new mechanism for proactive handling of impending faults, based on object migration. Upon receiving a warning of an imminent fault on a given processor, our runtime system efficiently migrates execution away from that processor.

The Colony project is also investigating and developing and all-Linux solution for Blue Gene. In addition to extending the spectrum of applications for Blue Gene/L, this activity will deliver a platform for studying extreme scalability of Linux.

The use of Linux as the operating system for Blue Gene compute nodes presents its own challenges. First and foremost, there is the issue of asynchronous and nondeterministic behavior, often observed in server-grade operating systems. How do we manage the many sources of asynchronous events in Linux (TLB misses, process/thread scheduler, I/O events) so that they do not impede scalability to tens of thousands of processors? Our recent work has focused on studying one particular source of asynchronous events: the TLB misses incurred by dynamic memory management. Figure 1 shows the performance obtained by Linpack running on 512 compute nodes for different page sizes. Performance degradation can be severe at usual Linux page sizes (4kB), but it is negligible for large pages.

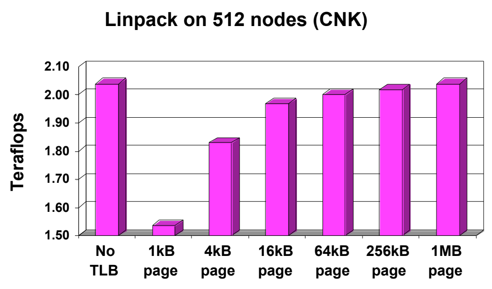

Figure 1: Performance for different page sizes.

We conclude that the asynchronous events caused by TLB management in Linux can be a source of performance degradation, but that we can address those problems by running with larger pages. Additional strategies are planned for other sources of asynchronous events including both process/thread scheduling and I/O events.

For further information on this subject contact:
Terry Jones, Coordinating PI
Lawrence Livermore National Laboratory
Email: trj@llnl.gov
Phone: 925-423-9834
Web: http://www.hpc-colony.org